Informe Final

Control de vibración de plataforma

(Sistemas embebidos)

Sistemas de tiempo real (2018)

Sánchez, Agustín (939/2)

Índice

1. Introducción

1.1 Consigna

La consigna presentada por la cátedra es la siguiente:

"Se debe controlar los niveles de una plataforma que contiene una máquina industrial.

Periódicamente se deben informar los valores de las mediciones. En caso de sobrepasar un límite se deben disparar las alarmas correspondientes e informar por algún mecanismo de comunicación a un servidor."

1.2 Motivación

Se busca realizar el proyecto con el fin de implementar los conocimientos otorgados por la cátedra acerca de los sistemas de tiempo real, llevando a cabo el control de eventos que ocurren en el mundo real y ejecutar acciones de respuesta para determinados estímulos con el entorno. En particular, se apunta a hacer uso de las herramientas que provee FreeRTOS para resolver la consigna.

2 Interpretación preliminar

Para dar lugar a la propuesta que satisfaga la consigna del proyecto, se la analizó y dividió en sus principales partes: sensores, medición, comunicación con el servidor y alarma.

2.1 Sensado

Se estipuló que se podría contar con un único sensor para abarcar la vibración general de la plataforma. Pero de todas formas se optó por utilizar al menos 3 sensores distribuidos por la plataforma para lograr sensar los 3 ejes dimensionale en los que podría vibrar y así lograr un monitoreo más eficiente de la vibración.

Primeramente se realizó un estudio de los posibles sensores que podían satisfacer las necesidades del proyecto, resumiendose en los siguientes dos sensores:

1) **Sensor 801s**: sensor con salida analogica ajustable de 3.3 - 5 volts. Posee un amplio rango de medición frente a vibraciones.
2) **Sensor SW420**: sensor con salida digital que se activa cuando la vibración medida supera un *threshold* definido manualmente por un potenciómetro. Menos sensible que el primero.

Es importante remarcar que para un monitoreo más minuciosos es deseable utilizar sensores de vibracion analogicos, pero tambien se pueden utilizar sensores con salida digital para acercarse a los mismos resultados de medición.

2.2 Medición

El mecanismo por el que se toman las mediciones de los sensores es el de *polling,* ya que se debe realizar una medición constante y de varios sensores simultáneamente. Además, se remarcó de antemano que se debería realizar pruebas para determinar cuál sería la frecuencia necesaria para lograr una medición eficiente, sin perder información importante.

2.3 Comunicación con un servidor

Primeramente es importante definir dónde residirá el servidor de monitoreo principal y el protocolo de comunicación que mantendrá con el sistema embebido. En esto último, se puede tomar dos ramas principales, alambrico o inalambrico, lo cual termina dependiendo de las decisiones que siguieron a continuación.

Para la implementación general del sistema se concluyó que las siguientes opciones son viables:

1) Arduino: cualquier variante de la placa de desarrollo Arduino, como lo son Arduino UNO, MEGA, WEMOS, etc. Dado que presentan 6 entradas analogicas de rango 0-5 volts, permite utilizar 3 sensores 801s repartidos por la plataforma. Con esta opción, la comunicación debe ser de forma serial (USB) hacia un programa corriendo en una PC cableada.

2) **ESP8266:** existe la posibilidad de utilizar un microcontrolador con la capacidad de comunicarse de forma inalámbrica, como puede ser el ESP8266. Dicha placa tiene conectividad Wi-Fi por lo que el sistema se desliga de tener que utilizar conexiones por cable. De esta forma el servidor puede vivir en cualquier computador y comunicarse con el sistema siempre y cuando se encuentre en la misma red. La desventaja de esta propuesta es que no presenta características tan robustas para la utilización industrial, por ejemplo se pueden presentar interferencias de radio frecuencia. Otra desventaja es que este tipo de placas solo poseen una entrada analogica y con un rango limitado de 0 - 1 volt, por lo que si se desea utilizar un sensor analogico se deberá utilizar un divisor de tensión para poder adaptar los rangos de voltajes. Si se desea utilizar más de un sensor, deberá optarse por utilizar sensores digitales.
3) **ESP32:** de forma similar a lo propuesto con el ESP8266, se podría utilizar la placa ESP32 la cual resuelve la limitación de tener 1 solo sensor analogico dado que posee 18 canales de ADC. Además no solo posee conectividad Wi-Fi sino que también Bluetooth.

2.4 Sistema de alarma

Dado que el objetivo del trabajo es en carácter de prototipado, se decidió utilizar simplemente notificaciones LED para cumplir el rol de alarma.

3 Propuesta final

Habiendo realizado el estudio preliminar, y evaluando los materiales que la cátedra y el alumno disponían, se concluyó en la siguiente propuesta.

En lo que respecta a sensores, se dispuso de 2 sensores digitales SW420, por lo que no se logró utilizar la opción óptima de sensores analógicos.

Se utiliza la placa de desarrollo Arduino Uno conectada a través de comunicación serial a una PC, donde vive el programa de monitoreo principal. Dicho programa tiene como objetivo proveer de una interfaz simple con el usuario para la visualización en tiempo real de los valores de vibración de cada sensor, como así permitir las configuraciones del sistema.

Dichas configuraciones son dos: el valor límite que al sobrepasarse activaría el sistema de alarma y el tiempo de sensibilidad de dicho límite.

Se implementa un simple indicativo del nivel de vibración con 3 LEDs, para que el operario de la máquina no tenga la necesidad de acceder al sistema informático.

También se plantea contar con un corte de seguridad automático. Una vez superado el límite máximo de vibración aceptable (por cierta cantidad de tiempo, pre-configurado), se simula un corte de seguridad de la alimentación de la maquinaria (representado con un LED) mediante un relay.

El programa principal cumple el rol de servidor web local a la red de la supuesta fábrica, para lograr exponer el sistema informático a los dispositivos conectados.

4 Descripción Hardware

4.1 Arduino UNO Rev3

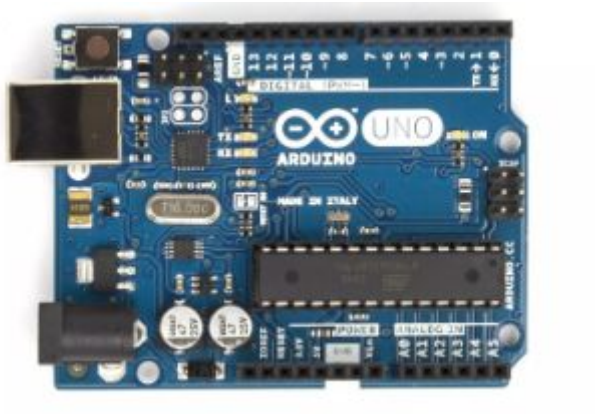

Figura 1. Microcontrolador Arduino Uno

El Arduino UNO es un microcontrolador *open-source* basado en el microchip ATmega328P. Dentro de sus características principales encontramos:

- 14 pines de entrada/salida digital
- 6 entradas analogicas
- Voltaje de operación de 5 Volts
- Frecuencia de clock de 16 MHz
- Conector USB

4.2 Sensor SW420

Este modulo está formado internamente por un resorte y un pequeño poste en su interior, provocando que cada vez que el sensor sea sometido a una vibración se genera un voltaje de salida. Dicha salida es comparada con un *threshold,* configurado por un potenciómetro, mediante un amplificador operacional embebido. De dicha comparación, se genera una salida digital donde HIGH representa una alteración.

Este sensor al ser digital, más que definirlo como un sensor de vibración, se lo debe definir como un detector de movimiento. Esto implica que su salida es binaria, movimiento o reposo, por lo que no nos permite medir magnitudes de vibración en primer medida.

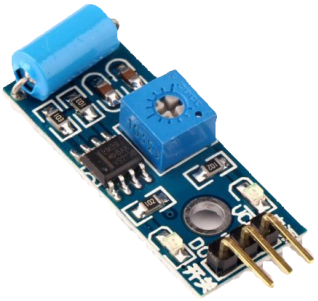

Figura 2. Sensor digital SW420

La solución que se encontró fue simular la medicion de valores analogicos mediante la implementación de un periodo de muestreo en los que se toman seguidas mediciones digitales. Luego se cuenta la cantidad de cambios en la señal digital pudiendo así traducir a una magnitud relativa de vibración.

$$0 < Vibración\ medida < 1$$

Por ejemplo, si se eligiese un periodo total de 20 ms, con mediciones cada 5 ms, los resultados serán los que se ven en la tabla 1 dependiendo de la cantidad de valores activos contados.

Tabla 1. Ejemplo de mediciones con 5 pasos.

Conteo	Vibración relativa
0	0
1	0.25
2	0.5
3	0.75
4	1

4.2.1 Ruido

Luego de realizar reiterados ensayos sobre estos sensores se determinó que contaban con ruidos de alta frecuencia debido a su sensibilidad por lo que se terminó optando por realizar un filtro pasa bajo digital para poder descartar dichos picos de ruido.

Para realizar los cálculos de las variables del filtro, se eligió por simularlo previamente logrando así los parámetros deseados. Dicho filtro fue escrito con *Processing*.

Se puede ver una demostración del funcionamiento del filtro en el siguiente link. A modo de explicación, la simulación cuenta de 3 filas. La primera es una señal autogenerada que representa a la señal original sensada. La segunda y tercer fila corresponden a las señales filtradas según su determinado buffer. Se ingresan diferentes valores del tamaño del buffer para encontrar valores óptimos.

4.3 Relay

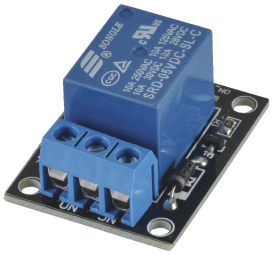

Figura 3. Modulo relay o relé.

Es un interruptor controlado mediante un circuito electrónico que mediante una bobina y un electroimán permiten el cierre y apertura de otros circuitos. Controla una alta tensión con un retorno de bajo voltaje, por lo que puede controlar el flujo de altas tensiones y corrientes con una operación de electricidad reducida y sin preocupación de "quemar" el controlador de relay, en este caso el Arduino.

Estas características lo convierte en perfecto candidato para realzar el corte de seguridad de alimentación de la maquinaria industrial, que por ahora se simula simplemente con un led encendido. Mediante una salida digital del microcontrolador se puede controlar la tensión de 220 Volts que consumiría la maquinaria.

4.5 Conexiones

En la figura 4 se presenta el diagrama de conexiones final donde se incluye el Arduino Uno junto a sus periféricos (se obviaron algunas líneas de alimentación para simplificar el diagrama).

Adicionalmente se presenta la figura 5 donde se puede ver la fotografía del prototipo terminado.

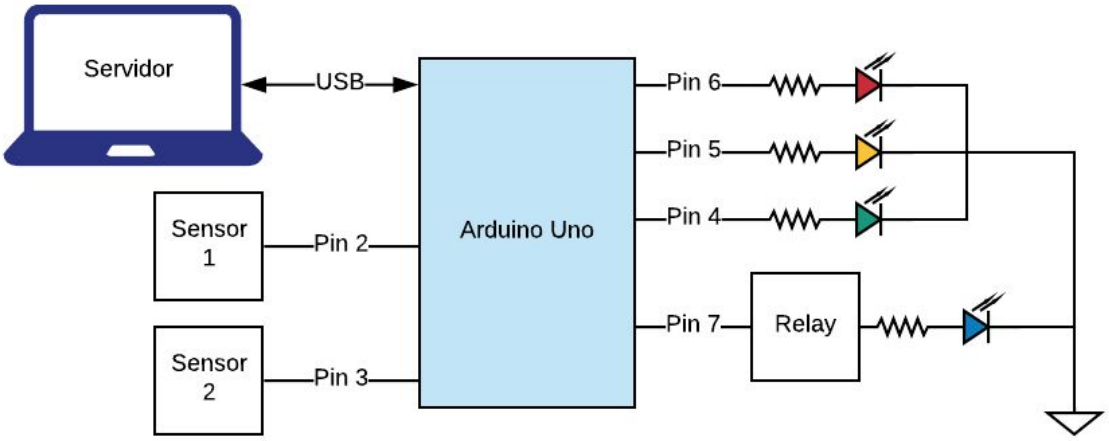

Figura 4. Esquema de conexiones básicas.

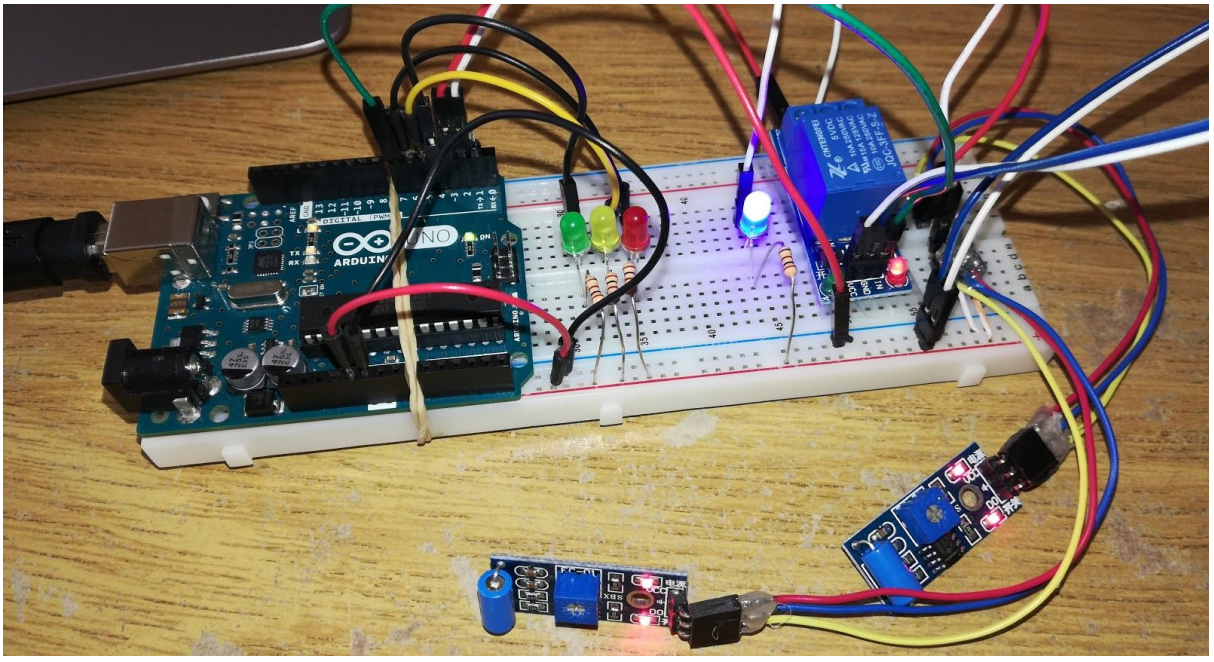

Figura 5. Fotografía de prototipo terminado.

5 Descripción del sistema general

La comunicación de la placa Arduino Uno con el sistema informático se realiza mediante comunicación serial, a la PC que hostea el servidor web local. Conociendo la IP de la computadora *servidor* el usuario podría acceder desde cualquier dispositivo de la red, sin necesidad de acceder únicamente a la computadora a la cual se encuentra conectado el Arduino.

Cabe destacar que de forma sencilla se podría ir más allá de lo implementado y exponer el servidor web de manera pública en Internet, logrando así la posibilidad del manejo remoto.

Una vez conectado al sistema de monitoreo, el usuario puede visualizar los valores actualizados sin necesidad de refrescar su navegador. También puede cambiar los parámetros de configuración del sistema de alarma mediante la interfaz prevista, y notando los cambios inmediatamente.

Si el sistema llega al punto crítico configurado, se “detiene” mediante el corte de seguridad generado por el relay que en el prototipo se puede observar mediante el LED azul asociado de alimentación.

A su vez para facilitar el trabajo del operario, se puede ver el nivel de vibración actual en el mismo sistema montado en la maquinaria, a través de los indicadores LED.

6 Descripción del SW

6.1 Arduino: Arquitectura FreeRTOS

FreeRTOS es un sistema operativo de tiempo real para sistemas embebidos, de código abierto y bajo continuo desarrollo, que permite correr tareas simultáneas con diferentes técnicas de sincronización y scheduling. Se optó por utilizar este framework con el propósito de implementar los conocimientos adquiridos durante la cátedra.

El diseño preliminar de la arquitectura del sistema embebido es la que se muestra en la figura 6.

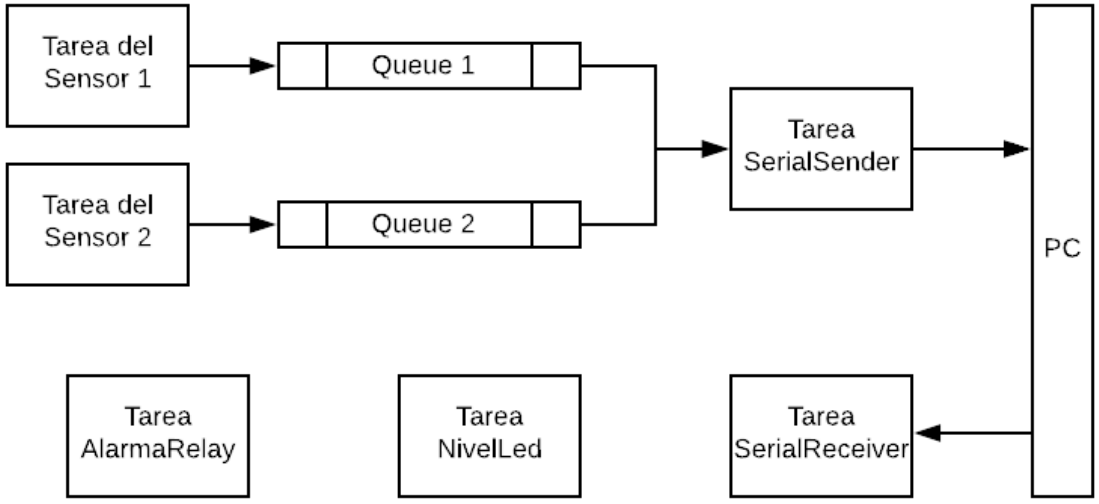

Figura 6. Arquitectura basa en tareas del sistema embebido.

Se pretendía contar con las siguientes tareas:

1) **Tarea Sensor 1/2**: encargada de iniciar las mediciones y aplicar el filtro digital diseñado, con su sensor correspondiente. Luego de obtener el valor de magnitud procesado, se lo coloca en una cola y se procede a tomar una nueva medición.
2) **Tarea SerialSender:** es la encargada de leer los valores generados por las tareas de los sensores, sincronizado y comunicados por una cola. Los empaqueta juntos para luego ser enviado de forma serial con el servidor web residente en la PC.
3) **Tarea SerialReceiver:** tarea con el fin de recibir los parametros de configuracion del sistema que el usuario podría transmitir a través de la PC, y realizar los ajustes de computación necesarios dentro del sistema embebido.

4) **Tarea NivelLed:** según los últimos valores procesados, esta tarea se encarga de actualizar los indicadores LED.
5) **Tarea AlarmaRelay:** tarea que monitorea el *threshold* establecido y en el caso de que sea necesario, activar el sistema de corte, osea accionar el relay.

La implementación de esta arquitectura generaba inconsistencia en la ejecución del sistema causando casi ,en todos los casos, que dejara de responder en su totalidad. Desafortunadamente luego de varios intentos con diferentes parámetros de configuraciones de las tareas y del scheduler, siempre intentando reducir el uso de memoria, no se logró resolver el problema.

Se concluyó utilizando solamente 4 tareas con las siguientes caracteristicas:

1) **Tarea Sensor1:** permanece igual. Prioridad 1.
2) **Tarea Sensor2:** permanece igual. Prioridad 1.
3) **Tarea SerialSender:** permanece igual. Prioridad 2.
4) **Tarea IDLE:** engloba todas las restantes. Prioridad 0.

A través de este nuevo diseño se obtuvo los resultados y comportamientos correspondientes. Ver Apéndice C, para analizar el código fuente. La inicialización del sistema es la siguiente:

```
138. void setup()
139. {
140.     Serial.begin(115200);
141.
142.     queue1 = xQueueCreate(1, sizeof(float));
143.     queue2 = xQueueCreate(1, sizeof(float));
144.
145.     if(queue1 == NULL)
146.         Serial.println("Error al crear cola 1");
147.     if(queue2 == NULL)
148.         Serial.println("Error al crear cola 2");
149.
150.     xTaskCreate(task_IDLE, "task_IDLE", 128, NULL, 0, NULL);
151.     xTaskCreate(task_sensor_1, "task_sensor_1", 128, NULL, 1, NULL);
152.     xTaskCreate(task_sensor_2, "task_sensor_2", 128, NULL, 1, NULL);
153.     xTaskCreate(task_sender, "task_sender", 128, NULL, 2, NULL);
154. }
155.
156. void loop() {}
```

6.1.1 PlatformIO

Es un ecosistema *open source* para el desarrollo multiplataforma de sistemas embebidos, con soporte de múltiples placas y frameworks diferentes. Permite su integración a editores de texto como Visual Studio Code. Por esas características se optó por utilizar esta herramienta, además de que el autor de este informe considera que agiliza en mayor medida el desarrollo del software a diferencia de su contraparte Arduino IDE.

La única diferencia a la hora de utilizarlo es que se debe importar la librería con las directivas de Arduino IDE a través de <Arduino.h> al principio del documento. Además permite la creación de librerías propias de una forma más sencilla e intuitiva. En este caso, se creó una para la implementación de la clase Sensor.cpp.

6.2 Máquina local: Servidor web

El servidor web viviría dentro de una máquina local a la fábrica y que se expone por WiFi para permitir el acceso de otros dispositivos de la misma red.

Para el desarrollo del servidor web se utilizó el *micro framework* para *Python* llamado *Flask.* Esta herramienta permite levantar un servidor con simples pasos y muy pocas líneas de código, por lo que fue la principal razón de su elección. Ver Apéndice A.

La arquitectura del servidor cuenta con dos principales hilos: el *core* del servidor propiamente dicho que se encarga de atender los *requests* y el hilo que se encarga de mantener una comunicación serial con el sistema embebido. A través de este canal se logra comunicar los valores recibidos de los sensores como así también modificar las configuraciones del sistema de monitoreo residentes en el microcontrolador.

Debido a los requisitos del proyecto, el servidor solo atiende solicitudes a la ruta principal.

6.2.1 GUI

La interfaz de usuario, que se presenta en la figura 7, fue desarrollada mediante la utilización del framework *Bootstrap* logran un diseño responsive.

Figura 7. Interfaz gráfica de usuario.

Del lado izquierdo se encuentra el panel de monitoreo, en el que se grafica en tiempo real, los valores medidos por los diferentes sensores. En este caso solo se encuentra con 2 sensores, pero es fácilmente escalable tanto por HW como por SW.

Del lado derecho se encuentra el panel de configuración, donde se puede manipular el nivel límite máximo que el sistema puede considerar seguro como así también su rango temporal de sensibilidad. En otras palabras, el tiempo que tiene que pasar estando por arriba del límite para que se active el sistema de seguridad de corte de alimentación representado por el relay y el LED.

6.2.2 WebSockets

El sistema de monitoreo debe mantener los gráficos de los sensores actualizados en tiempo real. De la forma tradicional en que funciona un sitio web, no se podría lograr, debido a que del lado del cliente se debería solicitar de forma constante el refrescado del sitio mediante protocolos *HTTP/HTTPS* generando cada vez un overhead de comunicación que va desde los cientos a los miles de milisegundos.

La forma en que se resolvió este problema fue mediante *Web-Sockets* que permite refrescar a una frecuencia de 50 ms y por solicitud del servidor (por el ingreso de nuevos datos) y no del usuario.

Ésto es posible, y sin ir en mucho detalle, gracias a que permite mantener una comunicación abierta full-duplex por un solo canal de TCP, por el tiempo que sea necesario, reduciendo así en gran medida el overhead de coneccion y desconeccion que tiene HTTP tradicional.

Por lo tanto, el servidor web al tener por primera vez un usuario abre una canal para el pasaje de mensajes mediante esta tecnología. Luego le transmite de forma continua los valores actualizados de los sensores que lee a través del hilo de comunicación serial con el Arduino. La comunicación del lado del cliente se logra gracias a la librería de WebSockets escrita en *JavaScript.*

No solo se utiliza este tipo de comunicación para actualizar los datos del usuario, sino que también para enviar los parámetros de configuración desde el usuario, hasta el servidor web (sin necesidad de refrescar) y hasta el Arduino mediante USB.

7 Resultados

7.1 Demostración monitoreo vibración

En el video que se puede ver en el siguiente link, se demuestra los resultados finales del proyecto. Se puede ver como al perturbarse los diferentes sensores, se observa de forma inmediata los cambios en la gráfica que representa la magnitud de la vibración.

Es importante destacar que en esta ocasión, los indicadores LEDs están configurados para funcionar únicamente con el primer sensor.

7.2 Demostración configuración del sistema

En la siguiente demostración que se encuentra en el siguiente link, se observa el funcionamiento de la configuración de los parámetros del corte automático de seguridad como también así su correcto funcionamiento.

El LED de color azul encendido indica que la maquinaria se encontraría en estado “encendido”. Luego de sobrepasar los valores máximos aceptables por 2 segundos, como fue configurado, se aprecia como el indicador se apaga, logrando así simular el apagado de la maquinaria entera.

La configuración por defecto provoca que una vez reducido el nivel de vibración, la maquinaria se vuelve a encender, fenómeno que también se puede apreciar en el video.

Apéndice A: Código fuente web server

El programa entero que se escribió para este proyecto se encuentra en el repositorio de github haciendo click en el link. Adicionalmente se adjunta el material en el presente apéndice.

A.1 app.py

```
1.  from flask import Flask, render_template
2.  from flask_socketio import SocketIO, emit
3.  from threading import Thread
4.  import time
5.  import eventlet
6.  eventlet.monkey_patch()
7.  import serial
8.  ser = serial.Serial('/dev/ttyACM0', 115200)
9.
10. app = Flask(__name__)
11. app.debug = True
12. app.config['SECRET_KEY'] = 'somesecretkey'
13. socketio = SocketIO(app)
14. thread = None
15.
16. def background_thread():
17.     while True:
18.         time.sleep(0.001)
19.         serial_line = ser.readline()
20.         if(len(serial_line) < 9):
21.             break
22.         sensor1 = serial_line[:4]
23.         sensor2 = serial_line[5:-2]
24.         print(sensor1 + "/" + sensor2)
25.         socketio.emit('value', {'value1': (float(sensor1)*10),
26.         'value2': (float(sensor2)*10)})
27.
28. @app.route('/')
29. def index():
30.     global thread
31.     if thread is None:
32.         thread = Thread(target=background_thread)
33.         thread.start()
34.     return render_template('index.html')
35.
36. @socketio.on('parametros')
37. def handle_parametros(json, methods=['GET', 'POST']):
38.     print("Segundos", json['segundos'])
39.     print("Threshold", json['threshold'])
40.     ser.write(b'S')
41.     ser.write([int(json['segundos'])])
42.     ser.write(b'T')
43.     ser.write([int(json['threshold'])])
44.
45. if __name__ == '__main__':
46.     socketio.run(app)
```

A.2 index.html

Del siguiente archivo solo se incluyen las partes más importantes.

```
1.  <script type="text/javascript"
    src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.3.6/socket.io.mi
    n.js"></script>
2.  <script type="text/javascript" charset="utf-8">
3.      var socket = io.connect('http://' + document.domain + ':' +
    location.port);
4.      socket.on('connect', function () {
5.          console.log('connected');
6.      });
7.      socket.on('value', function (data) {
8.          update_chart(data.value1, data.value2);
9.      });
10. </script>
11.
12. <script>
13.     var ctx = document.getElementById('myChart').getContext('2d');
14.     var chart = new Chart(...);
15.     function update_chart(v1, v2) {
16.         chart.data.datasets[0].data[0] = v1;
17.         chart.data.datasets[0].data[1] = v2;
18.         chart.update();
19.     }
20. </script>
21.
22. <script>
23.     var form = $('form').on('submit', function (e) {
24.         e.preventDefault();
25.
26.         let threshold = $('input#threshold-slider').val();
27.         let segundos = $('input#segundos').val();
28.
29.         console.log(threshold);
30.         console.log(segundos);
31.
32.         socket.emit('parametros', {
33.             threshold: threshold,
34.             segundos: segundos
35.         });
36.     });
37. </script>
```

Apéndice B: Filtro digital pasa bajo

Simulación del filtro digital para acondicionamiento de los sensores, escrito en Processing.

B.1 filter.pde

```
1.  int size = 200;
2.  float[] wave;
3.  float[] filtered1;
4.  float[] filtered2;
5.  float[] buffer1;
6.  float[] buffer2;
7.  int buffer_size1 = 1;
8.  int buffer_size2 = 1;
9.  float graph_canvas_height = 500;
10. float button_canvas_height = 60;
11. String button1 = "";
12. String button2 = "";
13. int button_selection = 0;
14. boolean overButton1 = false;
15. boolean overButton2 = false;
16. boolean overReset = false;
17.
18. void setup() {
19.    wave = new float[size];
20.    filtered1 = new float[size];
21.    filtered2 = new float[size];
22.
23.    button1 = str(buffer_size1);
24.    button2 = str(buffer_size2);
25.
26.    for(int i = 0; i < size; i++){
27.        wave[i] = random(1);
28.    }
29.    //poner un piso
30.    int param = 10;
31.    for(int i = 1; i < param; i++){
32.        wave[size-i] = 0;
33.    }
34.    //poner un techo
35.    for(int i = param; i < param*2; i++){
36.        wave[size-i] = 1;
37.    }
38.
39.    generate_filtered_waves();
40.    size(900, 560);
41.      textSize(15);
42. }
43.
44. void draw() {
45.    background(0);
46.    draw_graphs();
47.    draw_buttons();
48. }
49.
50.
```

```
51.  void draw_graphs(){
52.     fill(0,255,0);
53.     float w = 1.0 * width/size;
54.     float h_act = graph_canvas_height/3;
55.     for(int i = 0; i < size; i++){
56.         float h = h_act * wave[i];
57.         rect(i*w,h_act - h,w,h);
58.     }
59.     for(int i = 0; i < size; i++){
60.         float h = h_act * filtered1[i];
61.         rect(i*w,h_act * 2 - h,w,h);
62.     }
63.     for(int i = 0; i < size; i++){
64.         float h = h_act * filtered2[i];
65.         rect(i*w,h_act * 3 - h,w,h);
66.     }
67.     stroke(255);
68.     line(0, graph_canvas_height/3 * 1, width, graph_canvas_height/3 * 1);
69.     line(0, graph_canvas_height/3 * 2, width, graph_canvas_height/3 * 2);
70.     line(0, graph_canvas_height/3 * 3, width, graph_canvas_height/3 * 3);
71.     stroke(0);
72.  }
73.
74.  void draw_buttons(){
75.     //bg
76.     fill(75,75,75);
77.     rect(0, graph_canvas_height, width, button_canvas_height);
78.     //button1
79.     fill(255,255,255);
80.     textAlign(LEFT, CENTER);
81.     text("Buffer #1", 10, graph_canvas_height, 70, button_canvas_height);
82.     if(button_selection == 1)
83.         stroke(0,255,0);
84.     fill(255,255,255);
85.     rect(90,graph_canvas_height + 15,40,30);
86.     stroke(0);
87.     fill(0,0,0);
88.     textAlign(CENTER, CENTER);
89.     text(button1, 90, graph_canvas_height + 15, 40, 27);
90.
91.     //button2
92.     fill(255,255,255);
93.     textAlign(LEFT, CENTER);
94.     text("Buffer #2", 10+140, graph_canvas_height, 70,
    button_canvas_height);
95.     if(button_selection == 2)
96.         stroke(0,255,0);
97.     fill(255,255,255);
98.     rect(90+140,graph_canvas_height + 15,40,30);
99.     stroke(0);
100.    fill(0,0,0);
101.    textAlign(CENTER, CENTER);
102.    text(button2, 90+140, graph_canvas_height + 15, 40, 27);
103. }
104. void keyPressed() {
105.  if(button_selection == 0){
106.    return;
107.  }
```

```
 if(keyCode == ENTER){
   try {
       buffer_size1 = Integer.parseInt(button1);
   } catch (Exception e) {
       buffer_size1 = 1;
   }
   try{
       buffer_size2 = Integer.parseInt(button2);
   } catch (Exception e) {
       buffer_size2 = 1;
   }
   generate_filtered_waves();
   return;
 }
 if(button_selection == 1){
   if (keyCode == BACKSPACE) {
   if (button1.length() > 0) {
     button1 = button1.substring(0, button1.length()-1);
   }
 } else if (keyCode == DELETE) {
   button1 = "";
 } else if (keyCode != SHIFT && keyCode != CONTROL && keyCode != ALT) {
   button1 = button1 + key;
 }
 }
 if(button_selection == 2){
   if (keyCode == BACKSPACE) {
   if (button2.length() > 0) {
     button2 = button2.substring(0, button2.length()-1);
   }
 } else if (keyCode == DELETE) {
   button2 = "";
 } else if (keyCode != SHIFT && keyCode != CONTROL && keyCode != ALT) {
   button2 = button2 + key;
 }
 }
 }

void mouseMoved() {
 checkButtons();
}
 void mouseDragged() {
 checkButtons();
}



void mousePressed(){
 if(overButton1){
   button_selection = 1;
 }else if(overButton2){
   button_selection = 2;
 }else{
```

```
158.    button_selection = 0;
159.  }
160. }
161. void checkButtons() {
162.  if (mouseX > 90 && mouseX < 90+40 && mouseY > graph_canvas_height + 15
     && mouseY < graph_canvas_height + 15 + 30) {
163.    overButton1 = true;
164.  } else {
165.    overButton1 = false;
166.  }
167.  if (mouseX > 90+140 && mouseX < 90+140+40 && mouseY >
     graph_canvas_height + 15 && mouseY < graph_canvas_height + 15 + 30) {
168.    overButton2 = true;
169.  } else {
170.    overButton2 = false;
171.  }
172. }
173. void generate_filtered_waves(){
174.    buffer1 = new float[buffer_size1];
175.    buffer2 = new float[buffer_size2];
176.    for(int i = 0; i < buffer_size1; i++){
177.        buffer1[i] = 0;
178.    }
179.    for(int i = 0; i < buffer_size2; i++){
180.        buffer2[i] = 0;
181.    }
182.    for(int i = 0; i < size; i++){//filter
183.         //shift de buffers
184.         for(int j = 0; j < buffer_size1-1; j++){
185.             buffer1[j] = buffer1[j+1];
186.         }
187.         for(int j = 0; j < buffer_size2-1; j++){
188.             buffer2[j] = buffer2[j+1];
189.         }
190.         //calcular el valor filtrado
191.         buffer1[buffer_size1 - 1] = wave[i];
192.         float aux = 0;
193.         for(int j = 0; j < buffer_size1; j++){
194.                 aux += buffer1[j];
195.         }
196.         aux = aux / buffer_size1;
197.         filtered1[i] = aux;
198.         buffer2[buffer_size2 - 1] = wave[i];
199.         aux = 0;
200.         for(int j = 0; j < buffer_size2; j++){
201.                 aux += buffer2[j];
202.         }
203.         aux = aux / buffer_size2;
204.         filtered2[i] = aux;
205.    }//endfilter
206. }
```

Apéndice C: Código fuente Arduino

C.1 main.cpp

```
157. #include <Arduino.h>
158. #define INCLUDE_vTaskSuspend 1
159. #include <Arduino_FreeRTOS.h>
160. #include <queue.h>
161. #include <Sensor.h>
162.
163. #define PIN_SENSOR_1 2
164. #define PIN_SENSOR_2 3
165. #define PIN_LED_R 6
166. #define PIN_LED_Y 5
167. #define PIN_LED_G 4
168. #define PIN_RELAY 7
169. #define SAMPLE_STEP_PERIOD 16 //in miliseconds. el minimo es de 16 por
     cosas de FREERTOS
170. #define SAMPLE_PERIOD 16 * 3
171.
172. QueueHandle_t queue1;
173. QueueHandle_t queue2;
174. float value1, value2;
175. QueueHandle_t leds_sensor_1;
176.
177. char segundos = 5;
178. float threshold = 0.66;
179. void task_sensor_1(void *pvParameters)
180. {
181.    // Serial.println("Task 1");
182.    Sensor sensor(PIN_SENSOR_1);
183.    float value;
184.    int i;
185.    while (1)
186.    {
187.        sensor.measure_init();
188.        for (i = 0; i < SAMPLE_PERIOD; i += SAMPLE_STEP_PERIOD)
189.        {
190.            sensor.measure_tick();
191.            //fix feo
192.            delay(8);
193.            sensor.measure_tick();
194.            vTaskDelay(SAMPLE_STEP_PERIOD / portTICK_PERIOD_MS);
195.        }
196.        value = sensor.get_value();
197.        xQueueSend(queue1, &value, portMAX_DELAY);
198.    }
199. }
```

```
200. void task_sensor_2(void *pvParameters)
201. {
202.     // Serial.println("Task 1");
203.     float value;
204.     Sensor sensor(PIN_SENSOR_2);
205.     int i;
206.     while (1)
207.     {
208.         sensor.measure_init();
209.         for (i = 0; i < SAMPLE_PERIOD; i += SAMPLE_STEP_PERIOD)
210.         {
211.             sensor.measure_tick();
212.             //fix feo
213.             delay(8);
214.             sensor.measure_tick();
215.             vTaskDelay(SAMPLE_STEP_PERIOD / portTICK_PERIOD_MS);
216.         }
217.         value = sensor.get_value();
218.         xQueueSend(queue2, &value, portMAX_DELAY);
219.     }
220. }
221.
222. void task_sender(void *pvParameters)
223. {
224.     // Serial.println("Task Sender");
225.     while (1)
226.     {
227.         xQueueReceive(queue1, &value1, portMAX_DELAY);
228.         xQueueReceive(queue2, &value2, portMAX_DELAY);
229.         Serial.print(value1);
230.         Serial.print("/");
231.         Serial.println(value2);
232.     }
233. }
234. void task_IDLE(void *pvParameters)
235. {
236.     // Serial.println("Task IDLE");
237.     pinMode(PIN_LED_R, OUTPUT);
238.     pinMode(PIN_LED_Y, OUTPUT);
239.     pinMode(PIN_LED_G, OUTPUT);
240.     pinMode(PIN_RELAY, OUTPUT);
241.     unsigned long StartTime = 0;
242.     while (1)
243.     {
244.         // Serial.println("IDLE");
245.         if (Serial.available() > 0)
246.         {
247.             // read the incoming byte:
248.             char c = Serial.read();
249.             if(c == 'S'){
250.                 // Serial.print("Segundos ");
251.                 c = Serial.read();
252.                 segundos = c;
253.                 // Serial.println(c);
254.             }
255.             if(c == 'T'){
256.                 // Serial.print("Threshold ");
257.                 c = Serial.read();
```

```
258.                 threshold = c/100.0;
259.                 // Serial.println(c);
260.             }
261.         }
262.         //Leds de intensidad
263.         if (value1 > 0.05)
264.             digitalWrite(PIN_LED_G, LOW);
265.         else
266.             digitalWrite(PIN_LED_G, HIGH);
267.
268.         if (value1 > 0.33)
269.             digitalWrite(PIN_LED_Y, LOW);
270.         else
271.             digitalWrite(PIN_LED_Y, HIGH);
272.
273.         if (value1 > 0.66)
274.             digitalWrite(PIN_LED_R, LOW);
275.         else
276.             digitalWrite(PIN_LED_R, HIGH);
277.
278.         //relay
279.         if (value1 > threshold){
280.             if(StartTime == 0)//not measuring
281.                 StartTime = millis();
282.             else{//was measuring
283.                 // Serial.println((millis() - StartTime));
284.                 if((millis() - StartTime) > 1000 * segundos){
285.                     digitalWrite(PIN_RELAY, LOW);
286.                 }
287.             }
288.         }else{
289.             StartTime = 0;
290.             digitalWrite(PIN_RELAY, HIGH);
291.         }
292.     }
293. }
294. void setup()
295. {
296.     Serial.begin(115200);
297.
298.     queue1 = xQueueCreate(1, sizeof(float));
299.     queue2 = xQueueCreate(1, sizeof(float));
300.
301.     if(queue1 == NULL)
302.         Serial.println("Error al crear cola 1");
303.     if(queue2 == NULL)
304.         Serial.println("Error al crear cola 2");
305.
306.     xTaskCreate(task_IDLE, "task_IDLE", 128, NULL, 0, NULL);
307.     xTaskCreate(task_sensor_1, "task_sensor_1", 128, NULL, 1, NULL);
308.     xTaskCreate(task_sensor_2, "task_sensor_2", 128, NULL, 1, NULL);
309.     xTaskCreate(task_sender, "task_sender", 128, NULL, 2, NULL);
310. }
311.
312. void loop() {}
```

C.2 Sensor.h

```
#include <Arduino.h>
#define DIM 5
class Sensor
{
   uint8_t pin;
   uint8_t current_value;
   uint8_t iteration;
   uint8_t i;
   float aux;
   //para el filtro
   float previous_values[DIM];
 public:
   Sensor(uint8_t pin);
   void measure_tick();
   float get_value();
   void measure_init();
 private:
   float filter(float);
   void shift();
};
```

C.3 Sensor.cpp

```
#include <Sensor.h>

void Sensor::shift(){
   for(i = 0; i < DIM-1; i++){
       previous_values[i] = previous_values[i+1];
   }
}

float Sensor::filter(float value){
   shift();
   previous_values[DIM-1] = value;
   aux = 0;
   for(i = 0; i < DIM; i++){
       aux += previous_values[i];
   }
   aux = aux / DIM;
   return aux;
}

Sensor::Sensor(uint8_t p){
   pin = p;
   pinMode(pin, INPUT);
   measure_init();
   for(i = 0; i < DIM; i++){
       previous_values[i] = 0;
   }
}
void Sensor::measure_tick(){
   if(digitalRead(pin))
       current_value++;
   iteration++;
}
```

```
33.  float Sensor::get_value(){
34.     return filter(((float)current_value)/(iteration));
35.  }
36.
37.  void Sensor::measure_init(){
38.     current_value = 0;
39.     iteration = 0;
40.  }
```

Apéndice D: Links de interés

Resultados en videos:

- Filtro digital pasa bajo
- Demostracion sensado
- Demostracion configuracion

Repositorio del proyecto:

- Repositorio github